5.1 SERIAL VS. PARALLEL COMMUNICATION

Most logical operations and data processing occur in parallel on multiple bits simultaneously. Microprocessors, for example, have wide data buses to increase throughput. With wide buses comes a requirement for more wires to connect the logical elements in a system. The interconnection penalty increases as distances increase. Within a chip, the penalty is small, and wide buses are common. Implementing wide buses on a circuit board is also common because of the relatively short distances involved.

The economics and technical context of interconnect changes as soon as the distances grow from centimeters to meters to kilometers. Communication is primarily concerned with transporting data from one location to another rather than processing that information as it is carried on a wire. With distance comes the expensive problem of stringing a continuous wire between two locations. Whether the wire is threaded through a conduit between floors in an office, buried under the street between buildings, or virtually constructed via radio transmission to a satellite, the cost and complexity of connecting multiple wires is many orders of magnitude greater than on a circuit board. Serial communication is well suited to long distances, because fewer wires are used as compared to a parallel bus. A serial data link implies a single-wire medium, but there can be multiwire serial links as well.

Figure 5.1 illustrates several logical components in a serial data link. At either end are the sources and consumers of the data that operate using a parallel bus. A *transceiver* converts between a parallel bus and a serial stream and handles any link-level timing necessary to properly send and receive data. A *transducer*, or *modulator* in wireless links, converts between the medium's electromagnetic signaling characteristics and the transceiver's logic-level signals. Finally, a conductive path joins the two transducers. This path can be copper wire, glass fiber optic cable, or free space. These logical components may be integrated in arbitrary physical configurations in different implementations, so not all serial links will consist of three specific discrete pieces. Simple links may have fewer pieces, and complex links may have more.

The total cost of a data link is the sum of the cost of the transceiver/transducer subsystems at each end and the cost of the physical medium itself. A serial port on a desktop computer is inexpensive because of its relatively simple electronic circuits and because the medium over which it communicates, a short copper wire, is fairly cheap. In contrast, a satellite link is very expensive as a result of the greater complexity of the ground-based transmission equipment, the high cost of the satellite itself, and the licensing costs of using the public airwaves.

If only one bit is transferred per clock cycle in a serial link, it follows that either the serial bit clock has to be substantially faster than the parallel bus, or the link's *bandwidth* will be significantly

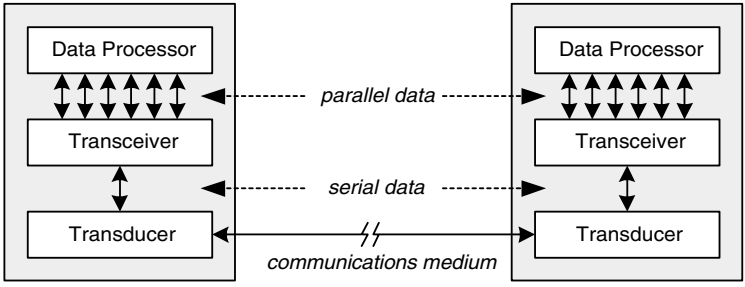

FIGURE 5.1 Components of a serial data link.

below that of the parallel bus. *Bandwidth* in a communication context refers to the capacity of the communications channel, often expressed either in bits-per-second (bps) or bytes-per-second (Bps). Serial links are available in a broad spectrum of bandwidths, from thousands of bits per second (kbps) to billions of bits per second (Gbps) and are stretching toward trillions of bits per second (Tbps)!

Most implementations in the kbps range involve applications where relatively small quantities of data are exchanged, so the cost of deploying an advanced data link is not justified. These serial links are able to run at low frequencies (several hundred kilohertz and below) and therefore do not require complex circuitry. Of course, some low-bandwidth data links can be very expensive if the medium over which they operate presents extreme technical difficulties, such as communicating across interplanetary distances. Implementations in the Gbps range serve applications such as high-end computer networks where huge volumes of data are carried. Such links are run at gigahertz frequencies and are relatively costly due to this high level of performance. Gigahertz serial transfer rates do not translate into similar logic clock frequencies. When a transceiver converts a serial data stream into a parallel bus, it contains the very high frequency complexity within itself. A 1-Gbps link requires only a 31.25 MHz clock when using a 32-bit data path.

5.2 THE UART

The *universal asynchronous receiver/transmitter* (UART) is a basic transceiver element that serializes a parallel bus when transmitting and deserializes the incoming stream when receiving. In addition to bus-width conversion, the UART also handles overhead and synchronization functions required to transport data. Data bits cannot simply be serialized onto a wire without some additional information to delineate the start and end of each unit of data. This delineation is called *framing*. The receiver must be able to recognize the start of a byte so that it can synchronize its shift registers and receive logic to properly capture the data. Basic framing is accomplished with a *start bit* that is assigned a logic state opposite to that of the transmission medium's idle state, often logic 1 for historical reasons. When no data is being sent, the transmission medium, typically a wire, may be driven to logic 1. A logic 0 start bit signals the receiver that data is on the way. The receiving UART must be configured to handle the same number of data bits sent by the transmitter. Either seven or eight data bits are supported by most UARTs. After seven or eight data bits have been captured following the start bit, the UART knows that the data unit has completed and it can resume waiting for a new start bit. One or more *stop bits* follow to provide a minimum delay between successive data units so that the receiver can complete processing of the current datum before receiving the next one.

Many UARTs also support some form of error detection in the form of a *parity bit*. The parity bit is the XOR of the data bits and is sent along with data so that it can be recalculated and verified at the receiver. Error detection is considered more important on a long-distance data link, as compared to on a circuit board, because errors are more prone over longer distances. A parity bit is added to each data unit, most often each byte, that tells the receiver if an odd or even number of 1s are in the data word. The receiver and transmitter must be configured to agree on whether even or odd parity is being implemented. Even parity is calculated by XORing all data bits, and odd parity is calculated by inverting even parity. The result is that, for even parity, the parity bit will be set if there are an odd number of 1s in the byte. Conversely, the parity bit will be cleared if there are an odd number of 1s present. Odd parity is just the opposite, as shown in Fig. 5.2.

Handshaking is another common feature of UARTs. Handshaking, also called *flow control*, is the general process whereby two ends of a data link confirm that each is ready to exchange data before the actual exchange occurs. The process can use hardware or software signaling. Hardware hand-